Database Project

Galinas Coffee

Making a Database for a Business

Christopher Castaneda

Jan-May 2021

Table of Contents

Part 1: Database Conceptual and Logical Design

Part 1, Question 1: Case Scenario

Case Scenario

Galinas Coffee is a small local business that resides in Galinas, CA. There are seven employees, and two of which are managers. The main product is coffee, but the store also sells a variety of pastries. The business' current struggle is an outdated inventory system, where they keep track of everything on an Excel sheet. Their goal is to digitize that and consolidate all information for easy access and reference, and give them the ability to query and report.

Business Needs

The growth of Galinas Coffee may be enhanced with a database to record and deliver information about store inventory, suppliers, employees, monthly expenses, and payroll.

Requirements

A database that holds and organizes important information about Galinas Coffee's operations.

Functions

Functions include recording inventory, making orders from all applicable suppliers, storing employee information, and tracking/reporting monthly operation expenses and payroll functions.

Activities

The data inputted in the database can be utilized to run queries and output results that are important to the business. Through the database, the store can forecast future customer orders, increase store productivity, and enhance the growth of Galinas Coffee in the community.

Entities

The main entities include Store, Employee, FriendsFamily, Inventory, Orders, and Supply.

Purpose

To grow Galinas Coffee by establishing a database that improves business operations through the organization of coffee and pastry orders as well as employee and supplier information for the store.

Part 1, Question 2: Business Rules

Each **supply/supplier** may ship one or more **orders**.
Each **order** must be received from one or more **supply/supplier**.

Each **employee** must be supervised by only one **employee**.
Each **employee** may supervise one or more **employees**.

Each **employee** must be employed either as an **hourly, seasonal,** or **manager employee**, but cannot be employed as more than one of those at once.

Each **store** must maintain one or more **inventory**, and each **inventory** must be maintained by a single **store**.

A **supply** must be of type of **Bakery, Coffee, Equipment,** or **Materials**, but cannot be more than one of these at a time.

Each **employee** must be employed at one and only one **store**, and each **store** must employ at least one **employee**.

Each **employee** may have one or more discounts for **FriendsFamily,** and **FriendsFamily** must use a single employee discount.

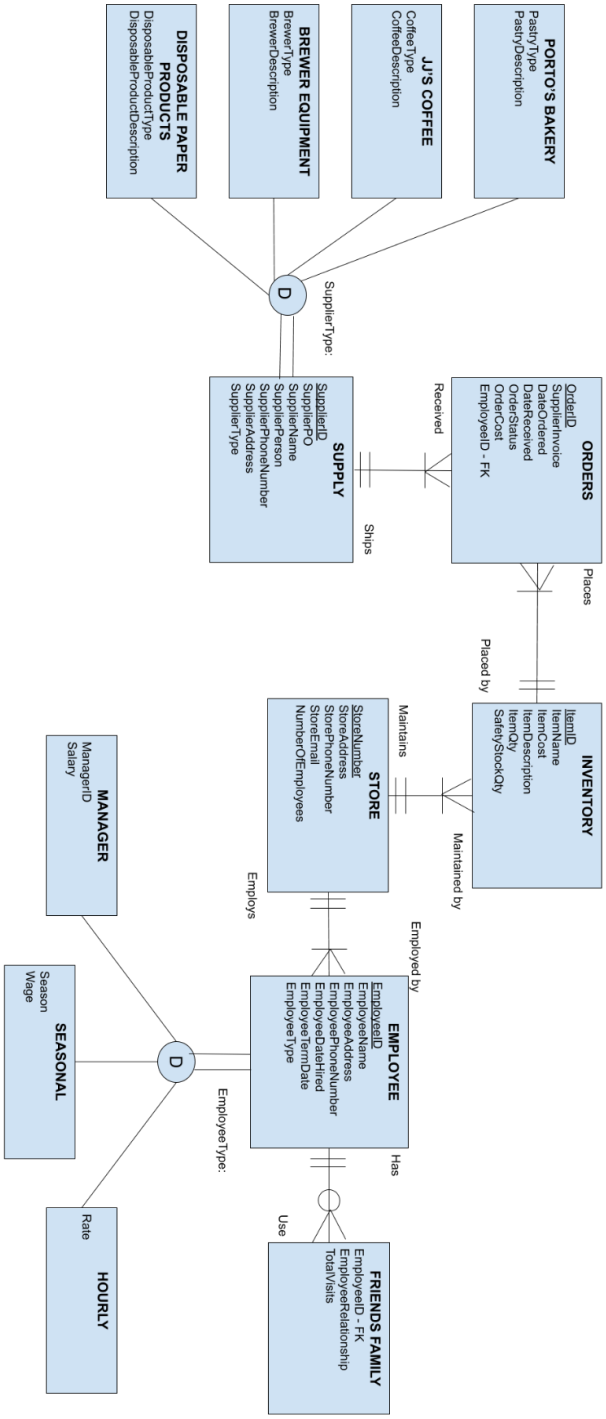
Part 1, Question 3: Relationship Matrix

	STORE	**EMPLOYEE**	**FRIENDS FAMILY**	**INVENTO-RY**	**ORDERS**	**SUPPLY**
STORE		Employed by		Maintained by		
EMPLO-YEE	Employs	Supervises/ Supervised by*	Uses			
FRIENDS FAMILY		Has				
INVENT-ORY	Maintain s				Places	
ORDERS				Placed by		Ships
SUPPLY					Received	

(*) This relationship is not indicated in the EERD, but is inferred by the relationship between Manager and Employees.

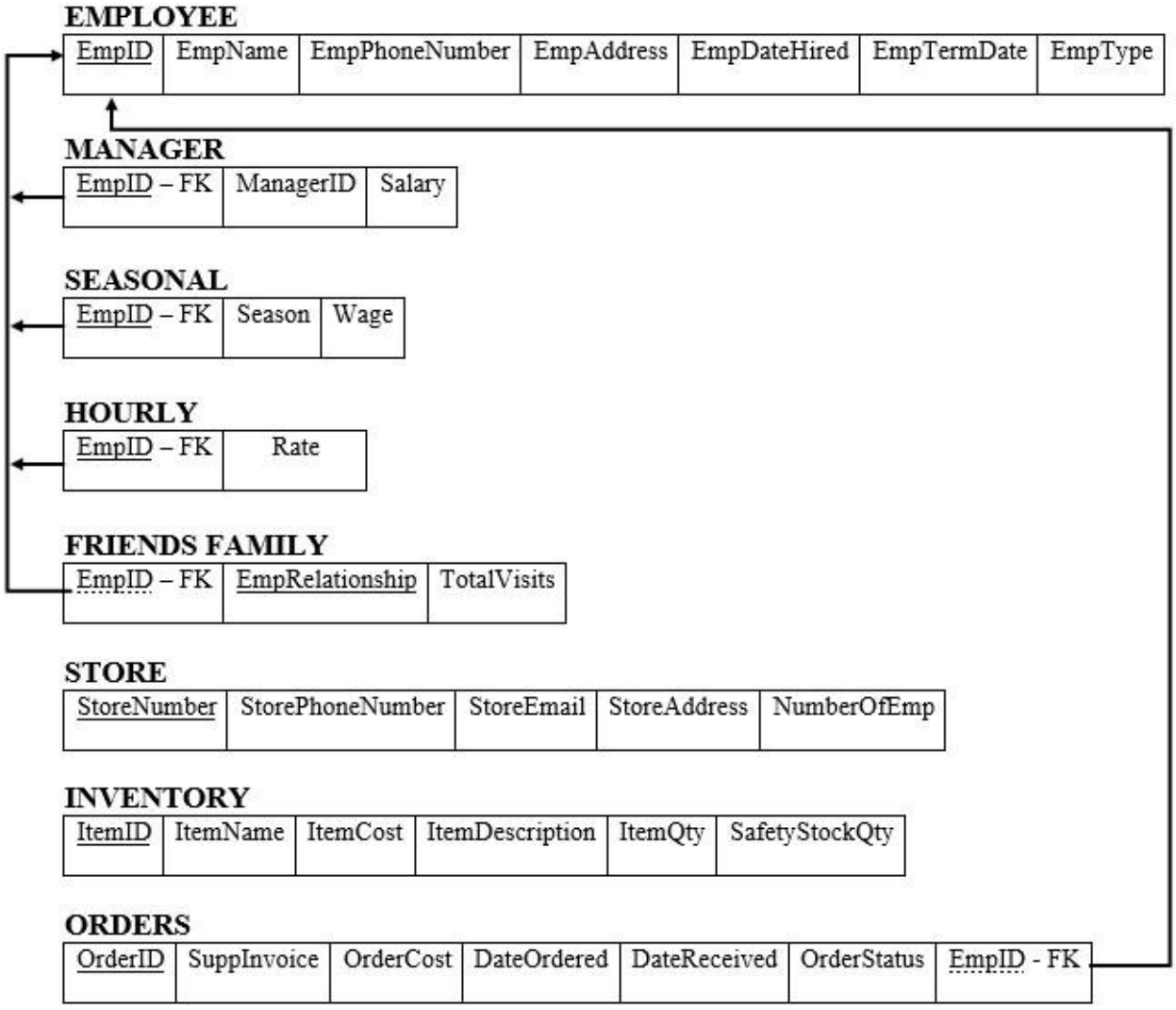
Part 1, Question 4: Entity Relationship Diagram (E[E]RD)

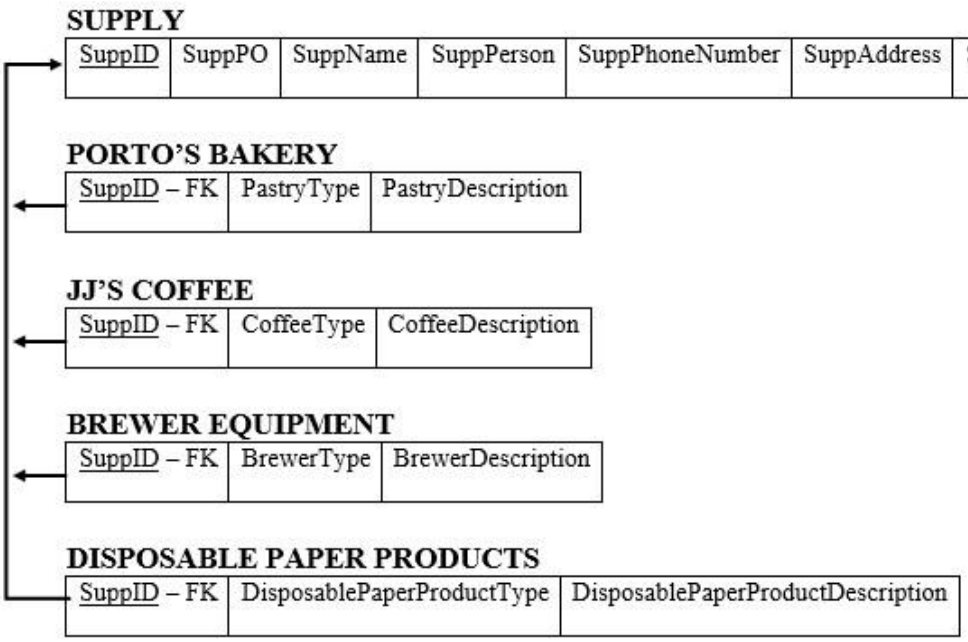
A bigger and clearer image of the EERD can be requested.



Part 1, Questions 5-6: Transferring EERD to Relations & Referential Integrity Constraints

Primary keys (PK) are underlined and foriegn keys (FK) are *dotted-underlined*. "Employee" type of attributes are abbreviated as "Emp," and "Supplier" type of attributes are abbreviated as "Supp" for the rest of "Part 1" questions in this report for text-fitting purposes. The database made and used in "Part 2" will have the appropriate naming conventions to run SQL queries. We use complete nouns as entity names to avoid confusion when the number of entities and attributes become expansive. We will also note that the name for each relation may be slightly modified in the final database creation, but they will retain the same purposes.





Arrows indicate referential integrity constraints. Referential integrity constraints are used to match each FK with a PK value in another relation. We have not encountered any foreign keys that should remain NULL. A bigger and clearer picture of the image can be requested.

Part 1, Question 7: Functional Dependency Diagram

Primary keys are underlined and foriegn keys are *italicized*, due to the nature of the application used for this diagram. Green arrows indicate full dependency; blue arrows indicate partial dependency; and yellow arrows indicate transitive dependencies. A full dependency is a functional dependency between two PKs. A partial dependency is a functional dependency derived from a PK with successive attributes. A transitive dependency is a functional dependency between the PK, and one or more nonkey attributes.

Part 1, Question 8: Normalization of Tables

All tables have been put into normal form, meaning that there are no composite or multivalued attributes. Due to the nature of Google Docs, we can not indicate FKs with dotted-underlines; instead, we italicized them for this part of the report.

EMPLOYEE

Emp ID	Emp First Name	Emp Last Name	Emp Phone Number	Emp Street	Emp City	Emp State	Emp Zip Code	Emp Date Hired	Emp Term Date	Emp Type

This relation is in 3rd normal form (*3NF*) because all values are atomic, and there are no multivalued attributes, partial dependencies, and transitive dependencies within this relation. Note that the EmployeeAddress of the Employee entity has been broken down into parts - Street, City, State, and ZipCode.

FRIENDS_FAMILY

EmpRelationship	TotalVisits	*EmpID* - FK

This relation is in *3NF* because all values are atomic, and there are no multivalued attributes, partial dependencies, and transitive dependencies within this relation.

STORE

Store Number	Store Phone Number	Store Email	Store Street	Store City	Store State	StoreZip Code	Number Of Emp

This relation is in *3NF* because all values are atomic, and there are no multivalued attributes, partial dependencies, and transitive dependencies.

INVENTORY

ItemID	ItemName	ItemCost	Item Description	ItemQty	Safety StockQty

This relation is in 2nd normal form (*2NF*) because all attributes are atomic values, and there are no multivalued attributes or partial dependencies. But there is a transitive dependency because the attribute "SafetyStockQty" is dependent on the attribute "ItemQty," and the user would be aware when the current item quantity is low, or reaching the level of the safety stock. Therefore, to change this table into *3NF*, ItemQty and SafetyStockQty will need a new table.

ORDERS

OrderID	SuppInvoice	Order Cost	Date Ordered	Date Received	Order Status	*EmpID* - FK

This relation is in *2NF* because all attributes are atomic values, and there are no multivalued attributes or partial dependencies, but there is a functional dependency. Note that OrderID and SuppID/SupplierID pair for the attribute SuppInvoice/SupplierInvoice. To make this table simple and in *3NF*, there should be a separate table for SuppInvoice/SupplierInvoice.

SUPPLY

Supp ID	Supp PO	Supp Name	Supp Person	Supp Phone Number	Supp Street	Supp City	Supp State	SuppZip Code	Supp Type

This relation is in *3NF* because all values are atomic, and there are no multivalued attributes, partial dependencies, and transitive dependencies.

Part 1, Question 9: Tables in Third Normal Form (3NF)

EMPLOYEE

Emp ID	Emp First Name	Emp Last Name	Emp Phone Number	Emp Street	Emp City	Emp State	Emp Zip Code	Emp Date Hired	Emp Term Date	Emp Type

FRIENDS_FAMILY

EmpRelationship	TotalVisits	*EmpID* - FK

STORE

Store Number	Store Phone Number	Store Email	Store Street	Store City	Store State	StoreZip Code	Number Of Emp

INVENTORY

ItemID	ItemName	ItemCost	Item Description

ITEM_STOCK

ItemQty	SafetyStockQty	*ItemID* - FK

ORDERS

OrderID	Order Cost	Date Ordered	Date Received	Order Status	*EmpID* - FK

INVOICES

OrderID	SuppID	SuppInvoice

SUPPLY

Supp ID	Supp PO	Supp Name	Supp Person	Supp Phone Number	Supp Street	Supp City	Supp State	SuppZip Code	Supp Type

Part 2: Database Implementation with SQL

Part 2, Question 1: Database Creation

```
CREATE TABLE "Employee_T"(
        "employeeID" NUMBER NOT NULL UNIQUE COLLATE NOCASE,
        "employeeFirstName" STRING NOT NULL UNIQUE,
        "employeeLastName" STRING NOT NULL UNIQUE,
        "employeeAddress"   STRING NOT NULL UNIQUE,
        "employeeCity"      STRING NOT NULL UNIQUE,
        "employeeState"     CHAR(2) NOT NULL UNIQUE,
        "employeeZip"       NUMBER NOT NULL UNIQUE,
        "employeePhone1"    STRING NOT NULL UNIQUE,
        "employeePhone2"    STRING UNIQUE,
        "employeeDOH"       DATE NOT NULL,
        "employeeTermDate" DATE,
        "employeeType"      CHAR(2) NOT NULL,
        PRIMARY KEY("employeeID")
);

CREATE TABLE "Manager_T" (
        "ManagerID"  INTEGER NOT NULL UNIQUE,
        "employeeSalary"    INTEGER NOT NULL
);

CREATE TABLE "Seasonal_T" (
        "employeeWage"      INTEGER NOT NULL,
        "employeeSeason"    TEXT
);

CREATE TABLE "Hourly_T" (
        "employeeRate"      INTEGER NOT NULL
);

CREATE TABLE "FriendsFamily_T" (
        "EmployeeID" NUMERIC,
        "EmployeeRelationships"     TEXT,
        "TotalVisits"   INTEGER,
        FOREIGN KEY("EmployeeID") REFERENCES "Employee_T"("employeeID")
);

CREATE TABLE "Store_T" (
        "storeNumber"       INTEGER UNIQUE,
        "storeAddress"      TEXT NOT NULL,
        "storeCity"     TEXT NOT NULL,
        "storeState"    TEXT NOT NULL,
```

```
        "storeZipCode"          NUMERIC NOT NULL,
        "storeEmployees"        INTEGER NOT NULL,
        "storePhoneNumber"  NUMERIC NOT NULL,
        “storeEmail”            NUMERIC NOT NULL,
        PRIMARY KEY("storeNumber")
);

CREATE TABLE "Inventory_T" (
        "itemID"        NUMERIC UNIQUE,
        "itemName"    TEXT NOT NULL,
        "itemQty"       INTEGER NOT NULL,
        "itemDescription"       TEXT NOT NULL,
        "itemSStockLevel"     INTEGER NOT NULL,
        PRIMARY KEY("itemID")
);

CREATE TABLE "Orders_T" (
        "orderID"       INTEGER NOT NULL UNIQUE,
        "supplierID"    NUMERIC NOT NULL,
        "employeeID" INTEGER NOT NULL,
        "orderDate"     NUMERIC,
        "orderReceived"         NUMERIC,
        "orderBackOrderItems"         INTEGER,
        PRIMARY KEY("orderID")
);

CREATE TABLE "Supply_T" (
        "supplierPO"   NUMERIC NOT NULL,
        "supplierID"    NUMERIC NOT NULL UNIQUE,
        "supplierName"          TEXT NOT NULL,
        "supplierContact"       TEXT NOT NULL,
        "supplierPhone1"        NUMERIC NOT NULL UNIQUE,
        "supplierPhone2"        NUMERIC,
        "supplierAddress"       TEXT NOT NULL,
        "supplierCity" TEXT NOT NULL,
        "supplierState"         TEXT NOT NULL,
        "supplierZip"   NUMERIC NOT NULL,
        "itemNumber" NUMERIC NOT NULL UNIQUE,
        "itemDescription"       TEXT NOT NULL,
        "supplierType"          TEXT NOT NULL,
        PRIMARY KEY("supplierID")
);

CREATE TABLE "PortosBakery_T" (
        "supplierPO"   STRING NOT NULL UNIQUE,
        "supplierName"          STRING NOT NULL UNIQUE,
```

```
        "itemNumber" INTEGER NOT NULL UNIQUE,
        "itemType"     STRING NOT NULL,
        "itemDescription"       STRING NOT NULL,
        "supplieStreet"          STRING,
        "supplierCity" STRING,
        "supplierState"          STRING,
        "supplierZip"  INTEGER,
        PRIMARY KEY("supplierPO")
);

CREATE TABLE "JJ'sCoffee_T" (
        "supplierPO"TEXT NOT NULL UNIQUE,
        "supplierName"TEXT NOT NULL UNIQUE,
        "itemNumber" TEXT NOT NULL UNIQUE,
        "itemType"TEXT NOT NULL,
        "itemDescription"TEXT NOT NULL,
        PRIMARY KEY("supplierPO")
);

CREATE TABLE "BrewerEquipment_T" (
        "supplierPO"   TEXT NOT NULL UNIQUE,
        "supplierName"         INTEGER NOT NULL UNIQUE,
        "itemNumber" TEXT NOT NULL UNIQUE,
        "itemType"     TEXT NOT NULL,
        "itemDescription"       TEXT NOT NULL,
        PRIMARY KEY("supplierPO")
);

CREATE TABLE "DisposablePaperProducts_T" (
        "DisposableProductType"      TEXT,
        "DisposableProductDescription"       TEXT
);
```

Part 2, Question 2: Data Insertion and SQL Scripts

The INPUT command is used to populate multiple rows.

INSERT INTO **Employee_T (employeeID, employeeFirstName, employeeLastname, employeeAddress, employeeCity, employeeState, employeeZip, employeePhone1, employeePhone2, employeeDOH, employeeTermDate, employeeType)**
Values (1, Arekg, Manash,123 1st St, Burbank, CA, 91506, 18182345678, NULL, 01012021, NULL, Hourly)
Values (2, Castaneda, Christopher, 19384 Birchwood Cr, Calabasas, CA, 90290, 8182882281, NULL , 01022021, NULL, Hourly)
Values (3, Darryn, Yost, 555 Mockingbird Lane, Anycity, CA, 91321, 17145551212, 16615551212, 01012021, NULL, Manager)
Values (4, Nomar, Garcia, 3920 Lindley St, Northridge, CA ,91326 , 8182892821, NULL , 01052021, NULL, Manager)
Values (5, Sarah, Le, 1920 W Clark Ave, Burbank, CA, 91506, 18180913519, NULL, 03012021, NULL, Seasonal)
Values (6, Tabba, Vince, 1929 W Steven Ave, Compton, CA, 02912, 8182922019, NULL, 04022021, NULL, Seasonal)
Values (7, Johnson, Clark, 29301 E Clemente St, Valencia, CA, 10292, 6619201831, NULL, 05292021, NULL, Hourly)

INSERT INTO **Manager_T (ManagerID, ManagerID, Salary)**
Values (3, MID3, 45000)
Values (4, MID4, 45000)

INSERT INTO **Seasonal_T (Season, Wage)**
Values (5, Summer, 14.50)
Values (6, Winter, 14.50)

INSERT INTO **Hourly_T (employeeID, employeeRate)**
Values (1, 17.00)
Values (2, 17.00)
Values (7, 17.00)

INSERT INTO **FriendsFamily_T (Employee ID,employeeRelationship, totalVisits)**
Values (1, cousin, 15)
Values (1, cousin, 30)
Values (2, friend, 31)
Values (2, friend, 2)
Values (6, family, 12)
Values (3, family, 21)
Values (4,friend,13)
Values (4,cousin,8)
Values (3, brother, 21)
Values (3, sister, 11)

Values (5, brother, 10)
Values (5, cousin, 25)

INSERT INTO **STORE_T (storeNumber, storeAddress, storeCity, storeState, storeZip, storeEmployees, storePhone)**
Values(001,123AppleStreet,Galinas,CA,45067(ZipCode),818-993-0211,galinascoffee1@gmail.com)

INSERT INTO **INVENTORY_T (Item ID, ItemName, ItemCost, ItemDescription, ItemQty, safetyStockQty)**
Values (001,Chocolate cookie,0.75,Pastry,30,10)
Values (002,Plain scone,0.75,Pastry,20,5)
Values (003,Spork,0.25,Utensil,50,25)
Values (004,Fork,0.25,Utensil,60,40)
Values (005,Spoon,0.25,Utensil,50,25)
Values (006,Knife,0.25,Utensil,30,15)
Values (007,Napkin,0.3,Utensil,80,40)
Values (008,Plain bagel,1.25,Pastry,20,5)
Values (009,Sugar cookie,0.75,Pastry,18,4)
Values (010,Sugar,0.4,Ingredients,100,70)
Values (011,Butter,0.8,Ingredients,150,100)
Values (012,Milk,0.6,Ingredients,60,50)
Values (013,Creamer,0.5,Ingredients,30,70)
Values (014,Blue Mountain,5,Ingredients,19,10)
Values (015,Blue Mountain (D),4.75,Ingredients,19,10)
Values (016,Evaporated milk,0.75,Ingredients,22,11)
Values (017,Oatmeal cookie,0.75,Pastry,8,2)
Values (018,Coffee cake,0.9,Pastry,6,2)
Values (019,Banana bread,0.9,Pastry,15,4)
Values (020,Danish,0.9,Pastry,14,5)
Values (021,Large cup,0.3,Cup,65,20)
Values (022,Medium cup,0.28,Cup,82,21)
Values (023,Small cup,0.26,Cup,33,58)
Values (024,Paper straw,0.25,Straw,90,59)
Values (025,Plastic straw,0.75,Straw,32,11)
Values (026,Metal straw,5,Straw,90,45)

INSERTS INTO **ORDERS_T (orderID, supplierIDE, employeeID, orderDate, orderReceived, orderBackOrderItems)**
Values (001,S01,02,01012021, 01032021,0)
Values (002,S04,05,01012021, 01032021,5)
Values (003,S02,03,01012021, 01032021,0)
Values (004,S01,01,02102021, 02122021,0)
Values (005,S02,04,02152021, 02172021,3)
Values (006,S01,01,02202021, 02222021,7)
Values (007,S04,03,03102021, 03122021,0)

Values (008,S02,04,03152021, 03172021,1)
Values (009,S03,03,03202021, 03222021,0)
Values (010,S01,02,03272021, 03292021,2)

INSERT INTO **SUPPLY_T (SupplierID, SupplierPO, SupplierName, SupplierPerson, SupplierPhoneNumber, SupplierAddress, supplierCity, supplierState, supplierZip, SupplierType)**

Values (S01, 1, Porto's Bakery, Kate, 18187007777, 10435 Lindley Ave, Northridge, CA, 91326, Bakery)
Values (S02, 2, JJ's Coffee, John, 13232927503, 14265 Terra Bella St , Pacoima, CA, 91304, Coffee)
Values (S03, 3, Brewer Equipment, Kevin, 12130295042, 10294 Candice Lane , Calabasas, CA, 90290, Equipment)
Values (S04, 4, Disposable Paper Products, Natalie, 18181039153, 102 Northrills Blvd, Las Vegas, NV, 50914, Materials)

INSERT INTO **BAKERY_T (PastryType, PastryDescription)**

Values (Cake,Coffee cake)
Values (Donut,Chocolate donut)
Values (Donut,Raspberry donut)
Values (Donut,Filled donut)
Values (Donut,Blueberry donut)
Values (Scone,Lemon scone)
Values (Scone,Raspberry scone)
Values (Cookie,Chocolate cookie)
Values (Scone,Plain scone)
Values (Bagel,Plain bagel)
Values (Cookie,Sugar cookie)
Values (Cookie,Oatmeal cookie)
Values (Bread,Banana bread)
Values (Bread,Danish)

INSERT INTO **JJ'sCoffee_T (CoffeeType, CoffeeDescription)**

Values (Blue Mountain, Regular)
Values (Columbian Grounds, Regular)
Values (Blue Mountain (D), Decaf)
Values (Sumatra, Decaf)
Values (Venice, Regular)
Values (Arabica, Regular)
Values (French Press, Decaf)
Values (Allegro, Regular)
Values (Blonde Vanilla, Regular)

INSERT INTO **BREWEREQUIPMENT_T (BrewerType, BrewerDescription)**

Values (Coffee, Brewer machine)
Values (Tea, Brewer machine)

Values (Dispenser, Brewer tool)
Values (Decanters, Brewer tool)
Values (Display cases, Brewer accessory)

INSERT INTO **Disposable Paper Products_T (disposableProductType, disposableProductDescription)**

Values (Cups, Small)
Values (Cups, Medium)
Values (Cups, Large)
Values (Straws, Metal)
Values (Straws, Plastic)
Values (Straws, Paper)
Values (Sleeves, Small)
Values (Sleeves, Medium)
Values (Sleeves, Large)
Values (Bags, Small)
Values (Bags, Large)
Values (Carriers,)
Values (Signage, small)
Values (Signage, large)

Part 2, Question 3: Query Questions and Statements

1. Show the employee IDs and names that work hourly.

```
SELECT Employee_T.EmployeeID, employeeFirstName, employeeLastName
FROM Employee_T, Hourly_T
WHERE Employee_T.employeeID = Hourly_T.employeeID
ORDER BY Employee_T.EmployeeID;
```

	employeeID	employeeFirstName	employeeLastName
1	1	Arekg	Manash
2	2	Christopher	Castaneda
3	7	Johnson	Clark

This query helps the business make different lists of those who are working under what type of employment. In this query, this would apply to hourly employees, or those working at a certain rate.

2. Show the employees that received orders in February, using JOINs.

```
SELECT Employee_T.employeeID, employeeFirstName, employeeLastName
FROM Employee_T
INNER JOIN Orders_T ON Employee_T.employeeID = Orders_T.EmployeeID
WHERE Orders_T.orderReceived BETWEEN 2012021 AND 2282021;
```

	employeeID	employeeFirstName	employeeLastName
1	1	Arekg	Manash
2	4	Nomar	Garcia
3	1	Arekg	Manash

This query helps the business find orders and who received/inspected them during a certain period. If there were issues with shipments and inventory during the month of February, then this query would also provide information to solve the issues.

3. Show the employee IDs and first names of those that have been used over 15 times for the friends & family discount (more than 15 visits), using a subquery.

```
SELECT Employee_T.employeeID, employeeFirstName
FROM Employee_T INNER JOIN FriendsFamily_T
ON FriendsFamily_T.employeeID = Employee_T.employeeID
WHERE Employee_T.EmployeeID IN
        (SELECT FriendsFamily_T.EmployeeID
        FROM FriendsFamily_T
        WHERE FriendsFamily_T.totalVisits > '15')
```

GROUP BY Employee_T.employeeID;

	employeeID	employeeFirstName
1	1	Arekg
2	2	Christopher
3	3	Darryn
4	5	Sarah

This query helps the business find the frequency of usage for the Friends & Family discount. If there are multiple employees having discounts used many times, then the store can expand the benefits program to customers who are loyal to the store and return often.

4. Show the item names and safety stock quantities from inventory that are below safety stock quantity.

```
SELECT ItemName, SafetyStockQty
FROM Inventory_T
WHERE ItemQty < SafetyStockQty;
```

	ItemName	SafetyStockQty
1	Creamer	70
2	Small cup	58

This query helps the business determine what items are below safety stock level, and allows them to make the necessary orders or adjust distribution of such items.

5. Show all employees that are managers, and that have received an order over $100, using JOINs.

```
SELECT DISTINCT Employee_T.employeeID, employeeFirstName,
employeeLastName
FROM Employee_T INNER JOIN Orders_T
ON Employee_T.EmployeeID = Orders_T.EmployeeID
WHERE Orders_T.orderCost > 100
AND Employee_T.employeeType = 'Manager';
```

	employeeID	employeeFirstName	employeeLastName
1	3	Darryn	Yost
2	4	Nomar	Garcia

This query helps the business find the orders received by the managers and the threshold of those order costs. This kind of query would also hold the employee accountable for processing such shipments into the store.

6. Find the supplier(s) that has delivered at least three orders, using COUNT.

```
SELECT SupplierName, count (orderID) as NumOfOrders
FROM Supply_T, Orders_T
WHERE Supply_T.SupplierID = Orders_T.SupplierID
GROUP BY SupplierName
HAVING COUNT (orderID) >= 3;
```

	SupplierName	NumOfOrders
1	JJ's Coffee	3
2	Porto's Bakery	4

This query identifies the frequency of orders delivered by supplier name. The store can use this data to plan orders for the future, either in bulk or per unit.

7. Find the average order cost for each supplier.

```
SELECT SupplierName, AVG (orderCost) AS 'SupplierAVG'
FROM Supply_T, Orders_T
WHERE Supply_T.SupplierID = Orders_T.supplierID
GROUP BY Supply_T.supplierID;
```

	SupplierName	SupplierAVG
1	Porto's Bakery	194.75
2	JJ's Coffee	325.666666666667
3	Brewer Equipment	3250.0
4	Disposable Paper Products	325.0

This query allows our business to see how much they are paying to each supplier on average. This would help with cost management as a business could use this information to compare other suppliers' costs and go with the supplier that allows for a bigger profit margin.

8. Show the total costs of each utensil currently in inventory.

```
SELECT ItemName, SUM (ItemCost * ItemQty) AS Total_Cost
FROM Inventory_T
WHERE ItemDescription = 'Utensil'
GROUP BY ItemName;
```

	ItemName	Total_Cost
1	Fork	15.0
2	Knife	7.5
3	Napkin	24.0
4	Spoon	12.5
5	Spork	12.5

This query finds the total costs of utensils currently in inventory. It can track supplies so that the store can be supported economically, project the future orders placed for such items, and record the frequency of item usage within the store.

9. Show the costs of the orders that every seasonal employee has made.

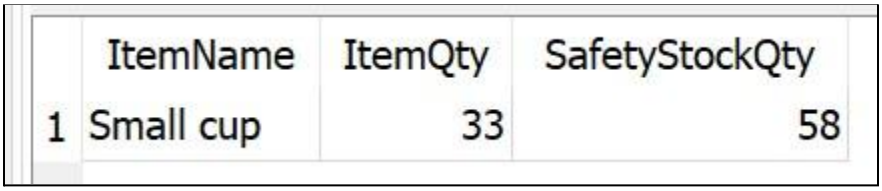
```
SELECT Orders_T.OrderCost, Employee_T.EmployeeID
FROM (Employee_T INNER JOIN Orders_T ON Employee_T.EmployeeID =
Orders_T.EmployeeID)
WHERE Employee_T.EmployeeType = 'Seasonal'
ORDER BY Orders_T.OrderID;
```

	orderCost	employeeID
1	500	5

This query identifies the particular employee type and whether they received an order. This would help with ensuring that orders were inspected and employees would be accountable for the process.

10. Show the item name, current quantity, and safety stock quantity of cups that are below safety stock level in inventory.

```
SELECT ItemName, ItemQty, SafetyStockQty
FROM Inventory_T
WHERE ItemDescription = 'Cup'
        AND ItemQty < SafetyStockQty
ORDER BY ItemID;
```



	ItemName	ItemQty	SafetyStockQty
1	Small cup	33	58

This query seeks to find a particular product that is below safety stock quantity. In this query, items with the description of 'Cup' were selected. The store should always avoid going below safety stock level, or they will fail to meet customers' demands.

Additional Files

The file for the database of Galinas Coffee is attached to this report. Additional files such as diagrams can be obtained upon request.